

```

scnPtr = img + (S.yOfsM+S.yWinM-1)*pixPerScn + S.xOfsM;
if (levels==8) {
    if (S.segmntMode==3) { /* directed acuity mode 3; pack pixels */
        /* S.xWinM and S.yWinM known to be mod 8 & mod 2, respectively */
        for (j=0; j<S.yWinM; j+=2, scnPtr-=2*pixPerScn) {
            pxlPtr = scnPtr;
            for (i=0; i<S.xWinM; i+=2, pxlPtr+=2) {
                bA = (int)*pxlPtr;
                bB = (int)*(pxlPtr+1);
                bC = (int)*(pxlPtr+pixPerScn);
                bD = (int)*(pxlPtr+pixPerScn+1);

                /* bit 0 is set if halftone (&1) */
                /* bit 1 is set if verticle (&2) */
                /* code for diagonals is same value in 224 and 28 locs */
                /* diagonals made only when bit d is blacker than a,b,c */
                vector = 0;
                if (bA&1) vector |= 8; /* halftone A */
                if (bB&1) vector |= 4; /* halftone B */
                if (bC&1) vector |= 2; /* halftone C */
                if (bD&1) vector |= 1; /* halftone D */
                switch (vector) {
                    case 0:if/*(((bD&224)<(bB&224))&&((bD&224)<(bC&224)))*/(0) {
                        byt = (bD&227) | ((bD>>3)&28); break;
                    } else {
                        byt = (bA&2)? (bA&227) | ((bB>>3)&28)
                                    : (bA&227) | ((bC>>3)&28);
                        byt = grad(byt); break;
                    }
                    case 1: byt = (bA&2)? (bA&227) | ((bB>>3)&28)
                                : (bA&227) | ((bC>>3)&28);
                        byt = grad(byt); break;
                    case 2: byt = (bB&2)? (bA&224) | ((bB>>3)&28) | (bB&3)
                                : (bA&224) | m28(bA) | (bB&3);
                        byt = grad(byt); break;
                    case 3: byt = (bA&2)? (bA&227) | ((bB>>3)&28)
                                : (bA&227) | m28(bA);
                        byt = grad(byt); break;
                    case 4: byt = (bC&2)? (bA&224) | m28(bA) | (bC&3)
                                : (bA&224) | ((bC>>3)&28) | (bC&3);
                        byt = grad(byt); break;
                    case 5: byt = (bA&2)? (bA&227) | m28(bA)
                                : (bA&227) | ((bC>>3)&28);
                        byt = grad(byt); break;
                    case 6: byt = bA&227 | m28(bA); break;
                    case 7: byt = bA&227 | m28(bA);
                        byt = grad(byt); break;

                    case 8:if/*(((bD&224)<(bB&224))&&((bD&224)<(bC&224)))*/(0) {
                        byt = (bD&227) | ((bD>>3)&28); break;
                    } else {
                        byt = (bD&2)? m224(bC) | ((bB>>3)&28) | (bB&3)
                                    : m224(bB) | ((bC>>3)&28) | (bB&3);
                        byt = grad(byt); break;
                    }
                    case 9: byt = (bB&2)? m224(bC) | ((bB>>3)&28) | (bB&3)
                                : m224(bB) | ((bC>>3)&28) | (bB&3);
                        byt = grad(byt); break;
                    case 10:
                    case 11: byt = m224(bB) | ((bB>>3)&28) | (2);
                        byt = grad(byt); break;
                    case 12:
                    case 13: byt = m224(bC) | ((bC>>3)&28);
                }
            }
        }
    }
}

```



```
    case 3: b3=8; break; /* 96 */
    case 2: b3=4; break; /* 64 */
    case 1: b3=0; break; /* 32 */
    case 0: b3=4; break; /* 0 */
}
b4 = byte&3;
b5 = b1|b3|b4;
return b5;
}

nt
28(byte)
nt byte;
if (byte&128) return 28; else return 0;

nt
24(byte)
nt byte;
if (byte&128) return 224; else return 0;
```

```

if (mcie==3) /* assemble das bytes into a,c,d pixels */
  srcPtr = srcStrt;
  ofsPtr = srcStrt + xwin*2; /* offset by one raster */
  nxtPtr = srcStrt + xwin*4; /* offset by two rasters */
  lsrPtr = srcStrt - xwin*4; /* offset by two rasters */
  for (i=0; i<ywin; i++) {
    for (j=0; j<xwin; j++) {
      byteP = *srcPtr;
      byteQ = *(srcPtr+2);
      byteR = *nxtPtr;
      byteS = *(nxtPtr+2);
      byteU = *lsrPtr; /* U & V have been unpacked last raster */
      byteV = *(srcPtr-2);
      crnrP = *(lslPtr-2);
      crnrQ = *(lslPtr+2);
      crnrR = *(nxtPtr-2);
      crnrS = *(nxtPtr+2);
      dg1PS = ((crnrP&1)&&(crnrS&1)); /* both PS corners lineart */
      dg1QR = ((crnrQ&1)&&(crnrR&1)); /* both QR corners lineart */
      /* lsb indicates lineart; next lsb indicates horizontal */
      if (byteP&1) { /* p is lineart - gen pqrs as lineart pixels */
        tmpP = (byteP&224);
        pxlp = tmpP|(tmpP>>3);
        if ( ((byteQ&2)&&(byteR&2)) /* one vert, the other horz */
            &&(byteQ&1)&&(byteR&1) /* and both sides are lineart */
            &&(byteU&1)&&(byteV&1) /* and four sides are lineart */
            &&((dg1PS) || (dg1QR)) ) { /* and either ps or qr lineart */
          /* if both diag lineart, replicate lowest along diag. */
          /* if only one diag lineart, replicate in dir of diag. */
          pxla = pxlp|1;
          tbot = byteP&28;
          pbot = tbot|(tbot<<3)|1;
          if ((dg1PS)&&(dg1QR)) { /* both diagonals lineart */
            /* replicate the lowest */
            if (pxla<pbot) { /* replicate a to d; b or c intorpolated */
              pxld = pxla;
              if (byteP&2) { /* p horz; bot of p goes to c; b intrp */
                tmpq = (byteQ&224);
                pxlq = tmpq|(tmpq>>3);
                pxlb = ((pxlp+pxlq)/2)|1;
                pxlc = pbot;
              } else { /* p vert; bot of p goes to b; c intrp */
                tmpq = (byteR&224);
                pxlr = tmpq|(tmpq>>3);
                pxlc = ((pxlp+pxlr)/2)|1;
                pxlb = pbot;
              }
            } else { /* replicate b to c; d intorpolated */
              pxlb = pbot;
              pxlc = pbot;
              tmpq = (byteQ&224);
              pxlq = tmpq|(tmpq>>3);
              tmpq = (byteR&224);
              pxlr = tmpq|(tmpq>>3);
              pxld = ((pxlq+pxlr)/2)|1;
            }
          } else { /* only one diagonal PS or QR is lineart */
            if (dg1PS) { /* only diagonal PS is lineart */
              /* replicate along PS direction */
              pxld = pxla;
              if (byteP&2) { /* p is horz */
                pxlc = pbot;
              }
            }
          }
        }
      }
    }
  }
}

```


